THE EMBODIMENTS OF THE INVENTION IN WHICH AN EXCLUSIVE PROPERTY OR PRIVILEGE IS CLAIMED ARE DEFINED AS FOLLOWS:

1. An optimizing compiler for compiling computer code, the optimizing compiler comprising,

   means for identifying a store operation at an original location in the computer code as a candidate for forward movement,

   means for identifying a location in the code into which the candidate store operation may be moved, at which identified location the store operation will not always be executed, and

   means for comparing the nearest preceding definition point for the variables in the store operation at both the original location and at the identified location to determine whether the candidate store operation may be correctly moved forward to the identified location and to specify the type of movement available in the potential optimization.

2. An optimizing compiler utilizing an intermediate representation of computer code to be compiled, the intermediate representation comprising blocks of computer code and a control flow graph, the optimizing compiler comprising

   traversing means for traversing the control flow graph in breadth-first order commencing at an exit block for the computer code,

   identifying means for identifying a target block into which a store operation reached in the traversal of the control flow graph may be moved,

   selecting means for determining whether the movement of the reached store operation to the target block may be correctly carried out and where the movement may be correctly carried out, adding an entry for the reached store operation and the target block to a store motion list, and

moving means for defining the movement of store operations on the store motion list to the respective locations of the target blocks on the store motion list.

3. The optimizing compiler of claim 2 in which the identifying means comprises means for identifying a target block by selecting a side node in the intermediate representation.

4. The optimizing compiler of claim 2 in which the intermediate representation further comprises a data flow graph, a dominator tree, and a post-dominator tree, and in which the identifying means further comprises

means for defining a set of reached uses blocks for the reached store operation, the set of reached uses blocks being defined by accessing the data flow graph,

means for traversing the dominator tree and for traversing the post-dominator tree to define the target block to be the first descendant, if any, of the reached store operation block which both

1. dominates each block in the set of reached uses blocks, and
2. does not post-dominate the reached store operation block.

5. The optimizing compiler of claim 2 in which the intermediate representation comprises a reaching defs table, and in which the selecting means comprises

means for comparing the reaching defs value for each load in the address expression of the reached store operation at its original location, with the reaching defs value for each load in the address expression of the reached store operation at the target block location, by accessing the reaching defs table, and

means for signalling the addition of an entry to the store motion list where the comparison of the said reaching defs values match.

6. The optimizing compiler of claim 5 in which the store motion list comprises data corresponding to a movement type for each entry in the list, and in which the moving means comprises,

means to traverse the store motion list,

means to determine the movement type for a store operation corresponding to an entry reached in the traversal of the store motion list, the movement type being determined by comparing the reaching defs values for each use in the right hand side expression of the reached store operation at its original location, with the reaching defs value for each use in the right hand side expression of the reached store operation at the target block location, by accessing the reaching defs table, and

where the full set of values match, setting the movement type of the reached entry to designate a move of the entire store operation,

where a subset of the values match, setting the movement type of the reached entry to designate a move of the partial right hand side of the store operation, storing in the store motion list the variables which are not able to be moved to the target block, and

where none of the values match, setting the movement type of the reached entry to designate a move of the left hand side of the store operation.

7. The optimizing compiler of claim 6, further comprising means for carrying out the movement of store operations in accordance with the movement type indicated in the store motion list, comprising

means for traversing the store motion list,

means for altering the intermediate representation of the computer code in accordance with the information in the entry in the store motion list reached during traversal of the list, comprising

means to move the entire representation of the store operation in the reached entry to the target block of the reached entry where the movement type for the reached entry designates a move of the entire store operation,

means to generate temporary variables corresponding to the variables which are not able to be moved to the target block stored in the reached entry, to generate one or more replacement store operations in the intermediate code, at the location of the store operation of the reached entry, and to generate one or more replacement store operations in the intermediate code, at the target block location of the reached entry, whereby the replacement store operations permit the substitution of the temporary variables in the replacement store operations and permit the partial movement of the store operation of the reached entry to be made without altering the correctness of the intermediate representation, where the movement type for the reached entry designates a move of the partial right hand side of the store operation and

means to generate temporary variables corresponding to the variables in the right hand side of the store operation of the reached entry, to generate one or more replacement store operations in the intermediate code, at the location of the store operation of the reached entry, and to generate one or more replacement store operations in the intermediate code, at the target block location of the reached entry, whereby the replacement store operations permit the substitution of the temporary variables in the replacement store operations of the reached entry, and the inclusion of the replacement store operations to be made without altering the correctness of the intermediate representation, where the movement type for the reached entry designates a move of the left hand side of the store operation.

8. The optimizing compiler of claim 7, further comprising means for updating a data flow graph in the intermediate representation to reflect any changes to the intermediate representation made in moving a store operation.

9. The optimizing compiler of claim 2, in which the intermediate representation of the code includes a tree representation and in which the traversal of the tree representation of code is carried out in a backward traversal order.

10. An optimizing compiler utilizing an intermediate representation of computer code to be compiled, the intermediate representation comprising blocks of computer code represented in tree format, a data flow graph, a dominator tree, a post-dominator tree, a reaching defs table, and a control flow graph, the optimizing compiler comprising

traversing means for traversing the control flow graph in breadth-first order commencing at an exit block for the computer code, and for traversing the tree representations of the code in reverse order,

identifying means for identifying a target block into which a store operation reached in the traversal of the control flow graph may be moved, identifying means comprising

means for defining a set of reached uses blocks for the reached store operation, the set of reached uses blocks being defined by accessing the data flow graph, means for traversing the dominator tree and for traversing the post-dominator tree to define the target block to be the first descendant, if any, of the reached store operation block which both

(a) dominates each block in the set of reached uses blocks, and

(b) does not post-dominate the reached store operation block,

selecting means for determining whether the movement of the reached store operation to the target block may be correctly carried out and where the movement may be correctly carried out, adding an entry for the reached store operation and the target block to a store motion list, the selecting means comprising,

means for comparing the reaching defs value for each load in the address expression of the reached store operation at its original location, with the reaching defs value for each load in the address expression of the reached store operation at the target block location, by accessing the reaching defs table, and

means for signalling the addition of an entry to the store motion list where the comparision of the said reaching defs values match, and

moving means for defining the movement of store operations on the store motion list to the respective locations of the target blocks on the store motion list, the moving means comprising,

means to traverse the store motion list,

means to determine the movement type for a store operation corresponding to an entry reached in the traversal of the store motion list, the movement type being determined by comparing the reaching defs values for each use in the right hand side expression of the reached store operation at its original location, with the reaching defs value for each use in the right hand side expression of the reached store operation at the target block location, by accessing the reaching defs table, and

where the full set of values match, setting the movement type of the reached entry to designate a move of the entire store operation,

where a subset of the values match, setting the movement type of the reached entry to designate a move of the partial right hand side of the store operation, storing in the store motion list the variables which are not able to be moved to the target block, and

where none of the values match, setting the movement type of the reached entry to designate a move of the left hand side of the store operation.

11. The optimizing compiler of claim 10, further comprising means for carrying out the movement of store operations in accordance with the movement type indicated in the store motion list, comprising

means for traversing the store motion list,

means for altering the intermediate representation of the computer code in accordance with the information in the entry in the store motion list reached during traversal of the list, comprising

means to move the tree representation of the store operation in the reached entry to the target block of the reached entry where the movement type for the reached entry designates a move of the entire store operation,

means to generate temporary variables corresponding to the variables which are not able to be moved to the target block stored in the reached entry, to generate one or more replacement store operations in the intermediate code, at the location of the store operation of the reached entry, and to generate one or more replacement store operations in the intermediate code, at the target block location of the reached entry, whereby the replacement store operations permit the substitution of the temporary variables in the replacement store operations and permit the partial movement of the store operation of the reached entry to be made without altering the correctness of the intermediate representation, where the movement type for the reached entry designates a move of the partial right hand side of the store operation and

means to generate temporary variables corresponding to the variables in the right hand side of the store operation of the reached entry, to generate one or more replacement store operations in the intermediate code, at the location of the store operation of the reached entry, and to generate one or more replacement store operations in the intermediate code, at the target block location of the reached entry, whereby the replacement store operations permit the substitution of the temporary variables in the replacement store operations of the reached entry, and the inclusion of the replacement store operations to be made without altering the correctness of the intermediate representation, where the movement type for the reached entry designates a move of the left hand side of the store operation.

12. The optimizing compiler of claim 11, further comprising means for updating the data flow graph in the intermediate representation to reflect any changes to the intermediate representation made in moving a store operation.

13. A method for determining a potential store forward optimization for the compilation of computer code, the method comprising the steps of,

identifying a store operation at an original location in the computer code as a candidate for forward movement, identifying a location in the computer code into which the candidate store operation may be moved, at which identified location the store operation will not always be executed, and

comparing the nearest preceding definition point for the variables in the store operation at both the original location and at the identified location to determine whether the candidate store operation may be correctly moved forward to the identified location and to specify the type of movement available in the potential optimization.

14. A method for determining a potential store forward optimization for the compilation of computer code, the compilation utilizing an intermediate representation of computer code to be compiled, the intermediate representation comprising blocks of computer code and a control flow graph, the method comprising the steps of

traversing the control flow graph in breadth-first order commencing at an exit block for the computer code,

identifying a target block into which a store operation reached in the traversal of the control flow graph may be moved,

determining whether the movement of the reached store operation to the target block may be correctly carried out and where the movement may be correctly carried out, adding an entry for the reached store operation and the target block to a store motion list, and

defining the movement of store operations on the store motion list to the respective locations of the target blocks on the store motion list.

15. The method of claim 14 in which the identifying step identifies a target block by selecting a side node in the intermediate representation.

16. The method of claim 14 in which the intermediate representation further comprises a data flow graph, a dominator tree, and a post-dominator tree, and in which the identifying step further comprises the steps of

defining a set of reached uses blocks for the reached store operation, the set of reached uses blocks being defined by accessing the data flow graph,

traversing the dominator tree and traversing the post-dominator tree to define the target block to be the first descendant, if any, of the reached store operation block which both

(a) dominates each block in the set of reached uses blocks, and

(b) does not post-dominate the reached store operation block.

17. The mehtod of claim 14 in which the intermediate representation comprises a reaching defs table, and in which the selecting step comprises the steps of

comparing the reaching defs value for each load in the address expression of the reached store operation at its original location, with the reaching defs value for each load in the address expression of the reached store operation at the target block location, by accessing the reaching defs table, and

signalling the addition of an entry to the store motion list where the comparison of the said reaching defs values match.

18. The method of claim 17 in which the store motion list comprises data corresponding to a movement type for each entry in the list, and in which the moving step further comprises the steps of,

traversing the store motion list,

determine the movement type for a store operation corresponding to an entry reached in the traversal of the store motion list, the movement type being determined by comparing the reaching defs values for each use in the right hand side expression of the reached store operation at its original location, with the reaching defs value for each use in the right hand side expression of the reached store operation at the target block location, by accessing the reaching defs table, and

where the full set of values match, setting the movement type of the reached entry to designate a move of the entire store operation,

where a subset of the values match, setting the movement type of the reached entry to designate a move of the partial right hand side of the store operation, storing in the store motion list the variables which are not able to be moved to the target block, and

where none of the values match, setting the movement type of the reached entry to designate a move of the left hand side of the store operation.

19. The method of claim 18, further comprising the step of carrying out the movement of store operations in accordance with the movement type indicated in the store motion list, comprising the following steps

traversing the store motion list,

altering the intermediate representation of the computer code in accordance with the information in the entry in the store motion list reached during traversal of the list, comprising the steps of moving the entire representation of the store operation in the reached entry to the target block of the reached entry where the movement type for the reached entry designates a move of the entire store operation,

generating temporary variables corresponding to the variables which are not able to be moved to the target block stored in the reached entry, generating one or more replacement store operations in the intermediate code, at the location of the store operation of the reached entry, and generating one or more replacement store operations in the intermediate code, at the target block location of the reached entry, whereby the replacement store operations permit the substitution of the temporary variables in the replacement store operations and permit the partial movement of the store operation of the reached entry to be made without altering the correctness of the intermediate representation, where the movement type for the reached entry designates a move of the partial right hand side of the store operation and

generating temporary variables corresponding to the variables in the right hand side of the store operation of the reached entry, generating one or more replacement store operations in the intermediate code, at the location of the store operation of the reached entry, and generating one or more replacement store operations in the intermediate code, at the target block location of the reached entry, whereby the replacement store operations permit the substitution of the temporary variables in the replacement store operations of the reached entry, and the inclusion of the replacement store operations is made without altering the correctness of the intermediate representation, where the movement type for the reached entry designates a move of the left hand side of the store operation.

20. The method of claim 19, further comprising the step of updating a data flow graph in the intermediate representation to reflect any changes to the intermediate representation made in moving a store operation.

21. The method of claim 14, in which the intermediate representation of the code includes a tree representation and in which the traversal of the tree representation of code is carried out in a backward traversal order.

22. A method for determining a potential store forward optimization for the compilation of computer code, the compilation utilizing an intermediate representation of computer code to be compiled, the intermediate representation comprising blocks of computer code represented in tree format, a data flow graph, a dominator tree, a post-dominator tree, a reaching defs table, and a control flow graph, the method comprising the steps of

traversing the control flow graph in breadth-first order commencing at an exit block for the computer code, and traversing the tree representations of the code in reverse order,

identifying a target block into which a store operation reached in the traversal of the control flow graph may be moved, by defining a set of reached uses blocks for the reached store operation, the set of reached uses blocks being defined by accessing the data flow graph, and traversing the dominator tree and the post-dominator tree to define the target block to be the first descendant, if any, of the reached store operation block which both

(a) dominates each block in the set of reached uses blocks, and

(b) does not post-dominate the reached store operation block, determining whether the movement of the reached store operation to the target block may be correctly carried out and where the movement may be correctly carried out, adding an entry for the reached store operation and the target block to a store motion list, by

comparing the reaching defs value for each load in the address expression of the reached store operation at its original location, with the reaching defs value for each load in the address expression of the reached store operation at the target block location, by accessing the reaching defs table, and

signalling the addition of an entry to the store motion list where the comparison of the said reaching defs values match, and

defining the movement of store operations on the store motion list to the respective locations of the target blocks on the store motion list, by

traversing the store motion list,

determining the movement type for a store operation corresponding to an entry reached in the traversal of the store motion list, the movement type being determined by comparing the reaching defs values for each use in the right hand side expression of the reached store operation at its original location, with the reaching defs value for each use in the right hand side expression of the reached store operation at the target block location, by accessing the reaching defs table, and

where the full set of values match, setting the movement type of the reached entry to designate a move of the entire store operation, where a subset of the values match, setting the movement type of the reached entry to designate a move of the partial right hand side of the store operation, storing in the store motion list the variables which are not able to be moved to the target block, and

where none of the values match, setting the movement type of the reached entry to designate a move of the left hand side of the store operation.

23. The method of claim 22, further comprising steps for carrying out the movement of store operations in accordance with the movement type indicated in the store motion list, comprising the steps of

traversing the store motion list,

altering the intermediate representation of the computer code in accordance with the information in the entry in the store motion list reached during traversal of the list, comprising

moving the tree representation of the store operation in the reached entry to the target block of the reached entry where the movement type for the reached entry designates a move of the entire store operation,

generating temporary variables corresponding to the variables which are not able to be moved to the target block stored in the reached entry, generating one or more replacement store operations in the intermediate code, at the location of the store operation of the reached entry, and generating one or more replacement store operations in the intermediate code, at the target block location of the reached entry, whereby the replacement store operations permit the substitution of the temporary variables in the replacement store operations and permit the partial movement of the store operation of the reached entry to be made without altering the correctness of the intermediate representation, where the movement type for the reached entry designates a move of the partial right hand side of the store operation and

generating temporary variables corresponding to the variables in the right hand side of the store operation of the reached entry, generating one or more replacement store operations in the intermediate code, at the location of the store operation of the reached entry, and generating one or more replacement store operations in the intermediate code, at the target block location of the reached entry, whereby the replacement store operations permit the substitution of the temporary variables in the replacement store operations of the reached entry, and the inclusion of the replacement store operations to be made without altering the correctness of the intermediate representation, where the movement type for the reached entry designates a move of the left hand side of the store operation.

24. The method of claim 23, further comprising the step of updating the data flow graph in the intermediate representation to reflect any changes to the intermediate representation made in moving a store operation.

25. A computer program product for the compilation of computer code, the computer program product comprising a computer usable medium having computer readable code means embodied in said medium, comprising computer readable program code means to carry out the method steps of claim 13.